COMP301 - HW2 - Sabuha KAYA

Q1)

a) - writing pseudo code:
- we have an array of size n → A[n]
- Smallest value = minVal

```
int A[n];
int minValue=0;
int minIndex;
for( i=0 , i<n , i++ )
    minIndex = i
    minValue = A[i]
    for( j= i+1 , j<n, j++)
        if( A[j] < A[minIndex]
            minIndex= j;
    swap( A[minIndx], A[i] )
        A[minIndex ]= A[i]
        A[i ]=minvalue;
```

b) This algorithm has two loop invariant
1) For the first for loop, array is sorted until i'th point
2) For the second inside loop, the minimum value is the always minimum in the range of A[i] to A[j].

if we prove this part, we can give an example

A[5] → {1, 5, 4, 2, 3}

first we will sort the outside loop. → {1, 2, 3, 4, 5}
while doing that we will watch the above steps in algorithm →

always min
1 < 5 ✓
min = 1
1 < 4 ✓
mn = 1
min < 3 ✓
min = 1

1 is the min.
do the same for
5, 4, 2, 3
and get → 1, 2, 3, 4, 5 → algorithm works perfectly!

c) Since we have n values in array and we have 2 for loops the worst case is $O(n^2)$. In best case scenario we can assume that the given array is already sorted But it doesn't change the fact that we have to iterate through all of the loops. And that means we have best and worst case scenario same for selection sort algorithm.

Q2) In the question it says us to look at while loop.
In while loop we have

```
while :)o ßß A[:] ) key
        A[i+1] = A[i]
        i = i-1;
```

what this loop does is mainly 2 procedures
① we are having a linear search to search through backward to a sorted sub-array in order to put the key value to the appropriate location.

② we are shifting the values bigger than "key" to the end of the array in order to take the actual key to the exact place.

Binary search is → halving the size of remaining portion of the sequence each time.
so if we apply binary search to decrese the running time of insertion sort that won't work. Because binary search decreases the num. of comparisons. for first statement, we will have to shift element to the end of the array in order to put key to the required position. And shifting all of these elements will take $O(n)$ time, which means the running time of the insertion sort will be $\Theta(n^2)$.
So we CANNOT do that.
→ if we use doubly linked list it is also $\Theta(n^2)$ → Sorting → n
inserting → n → because we have to navigate the link structure to find the insertion point → so, $\Theta(n^2)$

Q3)

To apply recursion algorithm

Firstly → Sort the sub-array A[1...n-1]

Second → Then get the value n - find proper place and insert in the array

When we insert the first element there is no need to sort that's why it takes $\Theta(1)$ time. After 1'st one since there are n-1 elements the running time is $\Theta(n)$

So Running time = $\begin{cases} \Theta(1) & n=1 \\ T(n-1)+\Theta(n) & n>1 \end{cases}$

→ Normal insertion sort method

```
Insert(Arr, key)
keyVal = Arr[key]
idx = key-1
while idx>0 && Arr[idx] > keyVal
    Arr[idx+1] = Arr[idx]
    idx = idx-1
Arr[idx+1] = keyVal
```

→ When we write the recursive version:

```
insertion_sort_recursion(Arr, n)

if n>1
    insertion_sort_recursion(Arr, n-1)
        Insert(Arr, n)
```

worst
→ If the element we insert is smaller than all of the elements we will shift every number in array which means $\Theta(n)$ running time.

average
→ the element is smaller than half of the array → $\frac{n}{2}$ → $\Theta(n)$ again

This will be done for every n elements in recursion → which means running time becomes → $n * \Theta(n) = \Theta(n^2)$